

 **Draft** Standards Track: Interface

EIP-5792: Wallet Function Call API

Adds JSON-RPC methods for sending multiple function calls from the user's wallet, and checking their status

Authors Moody Salem (@moodysalem)

Created 2022-10-17

Discussion Link <https://ethereum-magicians.org/t/eip-5792-wallet-abstract-transaction-send-api/11374>

Table of Contents

- Abstract
- Motivation
- Specification
 - `wallet_sendFunctionCallBundle`
 - `wallet_getBundleStatus`
 - `wallet_showBundleStatus`
- Rationale
- Backwards Compatibility
- Security Considerations
- Copyright

Abstract

Defines new JSON-RPC methods for dapps to send batches of function calls from a user's wallet, as well as check on the status of those calls. The new methods are more abstract in regard to the underlying transactions than the existing transaction sending APIs to allow for differences between wallet implementations, e.g. smart contract wallets utilizing EIP-4337 or EOA wallets that support bundled transactions via EIP-3074. Dapps may use this more abstract interface to support different kinds of wallets without additional effort, as well as provide a better UX for sending bundles of function calls (e.g. EIP-20 `#approve` followed by a contract call).

Motivation

The current methods used to send transactions from the user wallet and check their status are `eth_sendTransaction` and `eth_getTransactionReceipt`.

These methods are keyed on the hash of the on-chain transaction, i.e.

`eth_sendTransaction` returns a transaction hash computed from the transaction parameters, and `eth_getTransactionReceipt` takes as one argument the transaction hash. When the transaction hash changes, for example when a user speeds up the transaction in their wallet, the transaction hash that the dapp is aware of becomes irrelevant. There is no communication delivered to the dapp of the change in transaction hash, and no way to connect the old transaction hash to the new one, except by the user account and transaction nonce. It is not trivial for the dapp to find all signed transactions for a given nonce and account, especially for smart contract accounts which usually store the nonce in a contract storage slot. This happens more frequently with smart contract wallets, which usually use a third party relaying service and automatically re-broadcast transactions with higher gas prices.

These methods also do not support sending multiple function calls related to a single action. For example, when swapping on Uniswap, the user must often call EIP-20 `#approve` before calling the Uniswap router contract to swap. The dapp has to manage a complex multi-step asynchronous workflow to guide the user through sending a single swap. The ideal UX would be to bundle the approve call with the swap call, and abstract the underlying approve function call from the user.

The current interface also does not work well for account abstracted wallets (e.g. EIP-4337 or EIP-3074), which often involve a relaying service to sign and broadcast the transaction that triggers the function calls from the user's wallet. In these cases the actual transaction hash may not be known at the time of user signing, but must still be returned by `eth_sendTransaction`. The transaction hash returned by `eth_sendTransaction` in these cases is unlikely to be relevant to the transaction hash of the included transaction. The existing interface also provides no way to delay the resolution of the transaction hash, since it is used as the key of the transaction tracked by the dapp. Dapps often link to the block explorer for the returned transaction hash, but in these cases the transaction hash is wrong and the link will not work.

Dapps need a better interface for sending batches of function calls from the user's wallet so they can interact with wallets without considering the differences between wallet implementations. These methods are backwards compatible with existing EOA wallets. EOA wallets may send bundles of function calls as individual transactions. The goal of the new interface is to be a more stable and flexible interface that enables a better user experience, while wallet implementations evolve over time.

Specification

Three new JSON-RPC methods are added. Dapps may begin using these methods immediately, falling back to `eth_sendTransaction` and `eth_getTransactionReceipt` when they are not available.

wallet_sendFunctionCallBundle

Requests that the wallet deliver a group of function calls on-chain from the user's wallet.

The wallet:

- MUST send these calls in the order specified in the request.
- MUST send the calls on the request chain ID.
- MUST stop executing the calls if any call fails
- MUST NOT send any calls from the request if the user rejects the request
- MAY revert all calls if any call fails
- MAY send all the function calls as part of one transaction or multiple transactions, depending on wallet capability.
- MAY reject the request if the request chain ID does not match the currently selected chain ID.
- MAY reject the request if the `from` address does not match the enabled account.
- MAY reject the request if one or more calls in the bundle is expected to fail, when simulated sequentially

wallet_sendFunctionCallBundle OpenRPC Specification

```
- name: wallet_sendFunctionCallBundle
  summary: Sends a bundle of function calls from the user wallet
  params:
    - name: Function calls
      required: true
      schema:
        type: object
        title: Send function call bundle request
        required:
          - chainId
          - from
          - calls
        properties:
          chainId:
            title: chainId
            description: Chain ID that these calls should be sent on
            $ref: '#/components/schemas/uint'
          from:
            title: from address
            description: The address from which the function calls should be sent
            $ref: '#/components/schemas/address'
          calls:
            title: calls to make
            description: The calls that the wallet should make from the user's address
            type: array
```

```

minItems: 1
items:
  title: function call
  description: A single function call
  type: object
  required:
    - gas
    - data
  to:
    title: to address
    description: The address that is being called
    $ref: '#/components/schemas/address'
  gas:
    title: gas limit
    description: The gas limit for this particular call
    $ref: '#/components/schemas/uint'
  value:
    title: value
    description: How much value to send with the call
    $ref: '#/components/schemas/uint'
  data:
    title: data
    description: The data to send with the function call
    $ref: '#/components/schemas/bytes'
result:
  name: Bundle identifier
  schema:
    type: string
    maxLength: 66

```

wallet_sendFunctionCallBundle Example Parameters

```

[
  {
    "chainId": 1,
    "from": "0xd46e8dd67c5d32be8058bb8eb970870f07244567",
    "calls": [
      {
        "to": "0xd46e8dd67c5d32be8058bb8eb970870f07244567",
        "gas": "0x76c0",
        "value": "0x9184e72a",
        "data": "0xd46e8dd67c5d32be8d46e8dd67c5d32be8058bb8eb970870f072445675058bb8e"
      },
      {
        "to": "0xd46e8dd67c5d32be8058bb8eb970870f07244567",
        "gas": "0xdefa",
        "value": "0x182183",
        "data": "0xfbadbaf01"
      }
    ]
  }
]

```

```

    }
  ]
}
]
```

wallet_sendFunctionCallBundle Example Return Value

The identifier may be the hash of the call bundle, e.g.:

```
"0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331"
```

The identifier may be a numeric identifier represented as a hex string, e.g.:

```
"0x01"
```

The identifier may be a base64 encoded string:

```
"aGVsbG8gd29ybGQ="
```

wallet_getBundleStatus

Returns the status of a bundle that was sent via `wallet_sendFunctionCallBundle`. The identifier of the bundle is the value returned from the `wallet_sendFunctionCallBundle` RPC. Note this method only returns a subset of fields that `eth_getTransactionReceipt` returns, excluding any fields that may differ across wallet implementations.

wallet_getBundleStatus OpenRPC Specification

```

- name: wallet_getBundleStatus
  summary: Sends a bundle of function calls from the user wallet
  params:
    - name: Bundle identifier
      required: true
      schema:
        type: string
        title: Bundle identifier
  result:
    name: Call status
    schema:
      type: object
      properties:
        calls:
          type: array
          items:
            title: call status
            description: Status of the call at the given index
            type: object
```

```
status:
  title: The current status of the call
  enum:
    - CONFIRMED
    - PENDING
receipt:
  type: object
  required:
    - success
    - blockHash
    - blockNumber
    - blockTimestamp
    - gasUsed
    - transactionHash
  properties:
    logs:
      type: array
      items:
        title: Log object
        type: object
        properties:
          address:
            $ref: '#/components/schemas/address'
          data:
            title: data
            $ref: '#/components/schemas/bytes'
          topics:
            title: topics
            type: array
            items:
              $ref: '#/components/schemas/bytes32'
    success:
      type: boolean
      title: Whether the call succeeded
    blockHash:
      title: The hash of the block in which the call was included
      $ref: '#/components/schemas/bytes32'
    blockNumber:
      title: The number of the block in which the call was included
      $ref: '#/components/schemas/uint'
    blockTimestamp:
      title: The timestamp of the block in which the call was included
      $ref: '#/components/schemas/uint'
    gasUsed:
      title: How much gas the call actually used
      $ref: '#/components/schemas/uint'
    transactionHash:
      title: The hash of the transaction in which the call was made
      $ref: '#/components/schemas/bytes32'
```

wallet_getBundleStatus Example Parameters

As with the return value of `wallet_sendFunctionCallBundle`, the bundle identifier may be any string of max length `66`.

It may be the hash of the call bundle:

```
[
  "0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331"
]
```

It may contain a numeric identifier as a hex string:

```
[
  "0x01"
]
```

It may be a base64 encoded string:

```
[
  "aGVsbG8gd29ybGQ="
]
```

wallet_getBundleStatus Example Return Value

```
{
  "calls": [
    {
      "status": "CONFIRMED",
      "receipt": {
        "logs": [
          {
            "address": "0xa922b54716264130634d6ff183747a8ead91a40b",
            "topics": [
              "0x5a2a90727cc9d000dd060b1132a5c977c9702bb3a52afe360c9c22f0e9451a68"
            ],
            "data": "0xabcd"
          }
        ],
        "success": true,
        "blockHash": "0xf19bbafd9fd0124ec110b848e8de4ab4f62bf60c189524e54213285e7f54",
        "blockNumber": "0xabcd",
        "blockTimestamp": "0xabcdef",
        "gasUsed": "0xdef",
        "transactionHash": "0x9b7bb827c2e5e3c1a0a44dc53e573aa0b3af3bd1f9f5ed03071b16"
      }
    },
    {

```

```
"status": "PENDING"
  }
]
}
```

wallet_showBundleStatus

Requests that the wallet present UI showing the status of the given bundle. This allows dapps to delegate the display of the function call status to the wallet, which can most accurately render the current status of the bundle. This RPC is intended to replace the typical user experience of a dapp linking to a block explorer for a given transaction hash.

- The wallet MAY ignore the request, for example if the wallet is busy with other user actions.
- The wallet MAY direct the user to a third party block explorer for more information.

wallet_showBundleStatus OpenRPC Specification

```
- name: wallet_showBundleStatus
  summary: Requests that the wallet show the status of the bundle with the given id
  params:
    - name: Bundle identifier
      required: true
      schema:
        type: string
        maxLength: 66
  result:
    name: Empty
    schema:
      type: "null"
```

wallet_showBundleStatus Example Parameters

```
[
  "0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331"
]
```

wallet_showBundleStatus Example Return Value

```
null
```

Rationale

Account abstracted wallets, either via EIP-3074 or EIP-4337 or other specifications, have more capabilities than regular EOA accounts.

The `eth_sendTransaction` and `eth_getTransactionReceipt` methods limit the quality of in-dapp transaction construction and status tracking. It's possible for dapps to stop tracking transactions altogether and leave that UX to the wallet, but it is a better UX when dapps provide updates on transactions constructed within the dapp, because dapps will always have more context than the wallets on the action that a set of calls is meant to perform. For example, an approve and swap might both be related to a trade that a user is attempting to make. Without these APIs, it's necessary for a dapp to represent those actions as individual transactions.

Backwards Compatibility

Wallets that do not support the following methods should return error responses to the new JSON-RPC methods. Dapps MAY attempt to send the same bundle of calls via `eth_sendTransaction` when they receive a not implemented call, or otherwise indicate to the user that their wallet is not supported.

Security Considerations

Dapp developers MUST treat each call in a bundle as if the call was an independent transaction. In other words, there may be additional untrusted transactions between any of the calls in a bundle. The calls in the bundle may also be included in separate, non-contiguous blocks. There is no constraint over how long it will take a bundle to be included. Dapps must encode deadlines in the smart contract calls as they do today. Dapp developers MUST NOT assume that all calls are sent in a single transaction.

Copyright

Copyright and related rights waived via CC0.

Citation

Please cite this document as:

Moody Salem (@moodyselem), "EIP-5792: Wallet Function Call API [DRAFT]," *Ethereum Improvement Proposals*, no. 5792, October 2022. [Online serial]. Available: <https://eips.ethereum.org/EIPS/eip-5792>.